Naval Research Laboratory
Washington, DC 20375-5000

NRL Memorandum Report 6243

BaRT Manual
Preliminary Version 2.0

N. HOTA

Jaycor, 1608 Spring Hill Road
Vienna, VA 22180-2270

C. L. RAMSEY AND L. B. BOOKER

Naval Center for Applied Research in Artificial Intelligence
Information Technology Division

June 22, 1988

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) NRL Memorandum Report 6243	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Research Laborarory 6b. OFFICE SYMBOL (If applicable) Code 5510	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Washington, DC 20375-5000	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING / SPONSORING ORGANIZATION Office of Naval Research 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) Arlington, VA 22217	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO: 62234N PROJECT NO: RE21-24 1-805 TASK NO: WORK UNIT ACCESSION NO:

11. TITLE (Include Security Classification)
BaRT Manual Preliminary Version 2.0

12. PERSONAL AUTHOR(S)
Ramsey, C.L., Booker, L.B. and Hota,* N.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Interim	FROM 1986 TO 1987	1988 June 22	43

16. SUPPLEMENTARY NOTATION
*Jaycor, 1608 Spring Hill Road, Vienna, VA 22180-2270

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Classification problem solving
Bayesian reasoning
Expert system
Uncertainty
Belief maintenance

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

BaRT is an inference engine which has been developed to aid in classification problem solving. This tool is belief maintenance component of an expert system shell currently under development. This inference engine uses Bayesian reasoning and can handle problems associated with incomplete and uncertain evidence. It has successfully been used to perform ship classification. This manual describes how to load the BaRT program and how to use all of the available commands. This manual also provides some theoretical background and some implementation details concerning BaRT.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Connie L. Ramsey	22b. TELEPHONE (Include Area Code) (202) 767-2877 22c. OFFICE SYMBOL Code 5510

DD Form 1473, JUN 86 *Previous editions are obsolete* SECURITY CLASSIFICATION OF THIS PAGE

S/N 0102-LF-014-6603

CONTENTS

BaRT MANUAL PRELIMINARY VERSION 2.0

1. Introduction

Many real world problems are associated with uncertainty; the evidence people observe which helps them to reason about some goal event is almost always uncertain and incomplete. Still, people make judgements based on this uncertain and incomplete evidence. These uncertain evidences can be combined in various ways to find the validity or strength of a hypothesis [6,8], and Bayesian probability theory is a *normative* theory that allows one to reason about and combine uncertainties. Pearl has devised a way to represent, reason about and combine uncertain evidences in a way that conforms to the tenets of probability theory, but avoids the disadvantages usually associated with probabilistic computations of belief [6]. BaRT is a Bayesian Reasoning Tool which implements Pearl's ideas. It has been implemented as an AI programming environment which can perform classification problem solving, and it has been used to classify ships [11]. In BaRT, a classification problem is represented as a network of hypotheses. The belief in each value of each hypothesis can change as new evidence lends support to (or takes support away from) certain values of the hypothesis.

The rest of this manual is organized as follows. Section 2 provides an overview of the theoretical background for this work. Section 3 explains how to use BaRT and provides an example. Sections 4 and 5 provide details concerning the implementation of this system.

This manual describes a preliminary version of a system which is under development. Later versions of BaRT will have greater capabilities, so any of the functions and capabilities described here are subject to change.

2. Belief Networks

Pearl's framework provides a method for hierarchical probabilistic reasoning in directed, acyclic graphs called belief networks. Each node in the network represents a discrete-valued hypothesis which describes an aspect of the domain, and each node contains information about both the current belief of each value of the hypothesis and the most probable instantiation of the hypothesis given the evidence available, called the belief* distribution. (The belief* distribution provides a way to determine how the various degrees of belief in hypotheses can be interpreted. Generating a coherent explanation

Manuscript approved April 5, 1988.

involves the simultaneous acceptance of a set of hypotheses, a requirement that goes beyond simply noting the degree of belief in any individual hypothesis. This means that the problem solver must make a *commitment* in categorical terms about the best way to instantiate each hypothesis variable based on the evidence available.) Each link between two nodes represents a direct causal dependence between two of the hypotheses, and the directionality of the link is from cause to manifestation. Each node contains a tensor[1] of probabilities conditioned on the states of the causal variables; this tensor quantifies the relationship between a node and its parents (causes). It is important to note that numbers used to quantify the relationship do not have to be probabilities. All that is required is that the tensor entries are correct relative to each other.

The belief updating scheme keeps track of two sources of support for belief at each node: the diagnostic support derived from the evidence gathered by descendants of the node and the causal support derived from evidence gathered by parents of the node. Diagnostic support (λ) provides the kind of information summarized in a likelihood ratio for binary variables. Causal support (π) is the analogue of a prior probability, summarizing the background knowledge lending support to a belief. These two kinds of support are combined to compute the belief at a node with a computation that generalizes the odds/likelihood version of Bayes' rule. Each source of support is summarized by a separate local parameter, which makes it possible to perform diagnostic and causal inferences at the same time. These two local parameters (λ and π), together with the tensor of numbers quantifying the relationship between the node and its parents, are all that is required to update beliefs. Incoming evidence perturbs one or both of the support parameters for a node. This serves as an activation signal, causing belief at that node to be recomputed and support for neighboring nodes to be revised. The revised support is transmitted to the neighboring nodes, thereby propagating the impact of the evidence. Propagation continues until the network reaches equilibrium. The overall computation assigns a belief to each node that is consistent with probability theory. Using a similar computation, similar supporting factors (π^* and λ^*) are used to find the belief* distribution. Section 4 provides details concerning the implementation of this belief network, and the equations for belief and belief* updating are presented in Appendix A. The reader is referred to Pearl [6] for more details about the theoretical framework of this belief maintenance system.

3. Using BaRT

The system is implemented in Portable Common Loops (PCL) on top of Common LISP. (Note that PCL is very similar to Common LISP Object System Specifications

1 A *tensor* is a mathematical object that is a generalization of a vector to higher orders. The order of a tensor is the number of indices needed to specify an element. A vector is therefore a tensor of order one and a matrix is a tensor of order two.

(CLOSS).) A graphic interface is provided on Symbolics and Suns showing the network (the nodes and their relations), the belief vectors, the belief* vectors, the support parameters, the dynamic propagation of beliefs (or belief*) as new evidence is obtained, and other related values.

To build the network for a particular problem, the user must provide the specific information about the nodes, the links, and the joint conditional probability tensors. This information can be entered into the system by using the graphic interface knowledge acquisition routines provided by BaRT or by declaring the information in a data file which is presented to the program. See the sections on *Running the Program* and more specifically on *Knowledge Acquisition* to use the graphic interface routines. If the user prefers to enter this information in a data file, he should see the following section entitled *Data Files.*

3.1. Data Files

The network and its nodes and links can be defined in a data file. First declare the network with the macro *make-net* as follows:

(make-net '*network-name*)

where

network-name is the name of the network.

Now the user must declare each node and link in the network using the macros *make-node* and *make-link.* Make-node takes one argument and some keyword arguments. The first argument is the name to be given to the node being created. Keyword arguments are given below, and text in italics should be replaced by the user. To create a node, use make-node as follows:

(make-node '*nodename*
:role *documentation-string*
:node-values '(*val1 val2 val3 ... valn*)
:prior '*prior-probability*
:condprob '*joint-conditional-probability-tensor*
:parents-ord '(*pname1 pname2 pname3 ... pnamen*))

where

documentation-string is a documentation string explaining the role of this hypothesis in the overall model. The default value for this is nil.

val1, val2..valn are the possible values the hypothesis can take. The default value for this is '(true false).

prior-probability is a list of the prior probabilities of the values of the hypothesis. This is only needed for the top nodes.

joint-conditional-probability-tensor is the tensor quantifying the relationship between the manifestation (child) node and all of its causal (parent) nodes. This is only needed for network nodes which have parents, and it is represented as a list of lists of the form

$$\begin{array}{l}((P[X_1 \mid U_{1_1}\ U_{2_1} \ldots U_{n_1}]\ P[X_2 \mid U_{1_1}\ U_{2_1} \ldots U_{n_1}] \ldots\ P[X_m \mid U_{1_1}\ U_{2_1} \ldots U_{n_1})\\ (P[X_1 \mid U_{1_1}\ U_{2_1} \ldots U_{n_2}]\ P[X_2 \mid U_{1_1}\ U_{2_1} \ldots U_{n_2}] \ldots\ P[X_m \mid U_{1_1}\ U_{2_1} \ldots U_{n_2})\\ \quad \ldots\ldots\\ (P[X_1 \mid U_{1_j}\ U_{2_j} \ldots U_{n_i}]\ P[X_2 \mid U_{1_j}\ U_{2_j} \ldots U_{n_i}] \ldots\ P[X_m \mid U_{1_j}\ U_{2_j} \ldots U_{n_i}]))\end{array}$$

where X is a node and U_1, U_2, ..., U_n are all of its parents,

$P[X_i \mid U_{1_j}, U_{2_k}, \ldots U_{n_l}]$ is the probability of $X = X_i$ given that $U_1 = U_{1_j}$, $U_2 = U_{2_k}$, ... $U_n = U_{n_l}$,

X_1, X_2, ... X_m are the possible values of X,

U_{1_1}, U_{1_2}, ... U_{1_j}, are the possible values of U_1,

U_{2_1}, U_{2_2}, ..., U_{2_j}, are the possible values of U_2, ...

and U_{n_1}, U_{n_2}, ... U_{n_i}, are the possible values of U_n.

pname1, *pname2*..*pnamen* are the ordered node names of the parents of this node. The joint conditional probability tensor entries are based on this order in that the node name of parent U_1 is *pname1*, etc. Note that the slot *parents-ord* is not needed if there is only one parent.

Similarly make-link is used to create a link:

```
(make-link 'link-name
    :tnode 'top-node-name
    :bnode 'bottom-node-name)
```

where

link-name is the name of the link from the top node A to the bottom node B represented by A -> B.

top-node-name is the top node or the causal node of the link: node A.

bottom-node-name is the bottom node or the manifestation node of the link: node B.

An example which shows how to create a network data file is provided at the end of Section 3.

3.2. Running the Program

A graphic interface has been developed for Symbolics and Suns. (In order to run the program without the graphic interface, the reader should see Appendix B.) To run BaRT, the user should load the system and then choose the appropriate commands. This is explained in detail below.

3.2.1. Loading the System Definitions and the System onto PCL

Symbolics: Get into a Common LISP environment which has PCL. From the LISP listener, load the system definitions by loading the file *bart-defsys.lisp* which is in the *src* subdirectory of the *bart* directory. Then load the system with the command *(bart-util::load-bart)*. Now, invoke the program by first pressing the *Select* key and then pressing the *Symbol*, *Shift* and *B* keys simultaneously.

Sun: Invoke suntools, and go to the *src* subdirectory of the *bart* directory. Then type *run-bart* from the shell; this loads the system definitions and the system. Change to package *bart-frame* with the command *(in-package 'bart-frame)*. Now, invoke the program with the command *(bart-command-loop)*. Note that there is no return prompt from this command.

3.2.2. Using the Command Menus

There are several command levels within BaRT. Selecting the commands from the command menu or choosing the nodes and links is performed similarly at all levels.

Choosing Nodes and Links

In order to select a node or a link for certain commands, the user must mouse-click on the node/link. To select a node, the user must **click left on a node**. To select a link on the Symbolics, the user must **click left on a link**. To select a link on the Sun, the user must **click middle on each node** which is connected to the link.

Selecting Commands

The available commands are activated by **mouse-clicking left** on them. (On the Symbolics, the user can also activate a command whose first letter is in brackets by typing the first letter of that command.) In addition, brief documentation for each command is provided on the Sun by **mouse-clicking right** on it.

3.2.3. Top-Level BaRT Windows and Commands

BaRT Windows

After loading the BaRT system, the whole screen consists of six windows: the title pane, the belief network display pane, the global system parameters pane, the command menu pane, the node/link information display pane, and the LISP interaction window. Figure 1 provides a sample screen display.

The title window lies across the top of the screen and consists of the heading *Bayesian Reasoning Tool (BaRT)* in boldface.

The belief network display pane is on the left hand side of the screen occupying a large portion of the screen. This pane is used to display the network which consists of nodes and their links. Some of the nodes in the network may be grayed (depending on whether the option to *compute benefit factors* is selected); the intensity of the grayness measures the entropy (uncertainty) in the belief values of each node in the network.

The global system parameters pane is on the top right hand side of the screen and consists of two lines. The slot on the first line contains the data file name (this will be empty before loading the data file). If the selected data file name is in boldface, then the network is in equilibrium; otherwise the network is not in equilibrium. This distinction is useful when propagating the effect of new evidence in the network in step mode, i.e., updating one node at a time. The second line consists of two flags: *step* and *debug*. These are boolean flags. If the string is in bold letters, then it indicates that the flag is set on; otherwise it is off. Step mode allows the user to see the network update one node at a time. Debug mode is not yet implemented.

The command menu pane is right below the global system parameters pane and consists of the commands in the top-level command loop. These are mouse sensitive and can be invoked by mouse-clicking left on them. All mouse-sensitive objects are highlighted when the mouse arrow is on the object's region.

The node/link information display pane is below the command menu pane and is used to present information about nodes/links in the network.

In the bottom right hand corner of the screen is the interaction window for normal interaction. This window is used for displaying messages or prompting for information, and the user can enter information here. LISP expressions can also be evaluated in this pane by first clicking on *eval* in the command menu pane and then typing the expression to the interaction window.

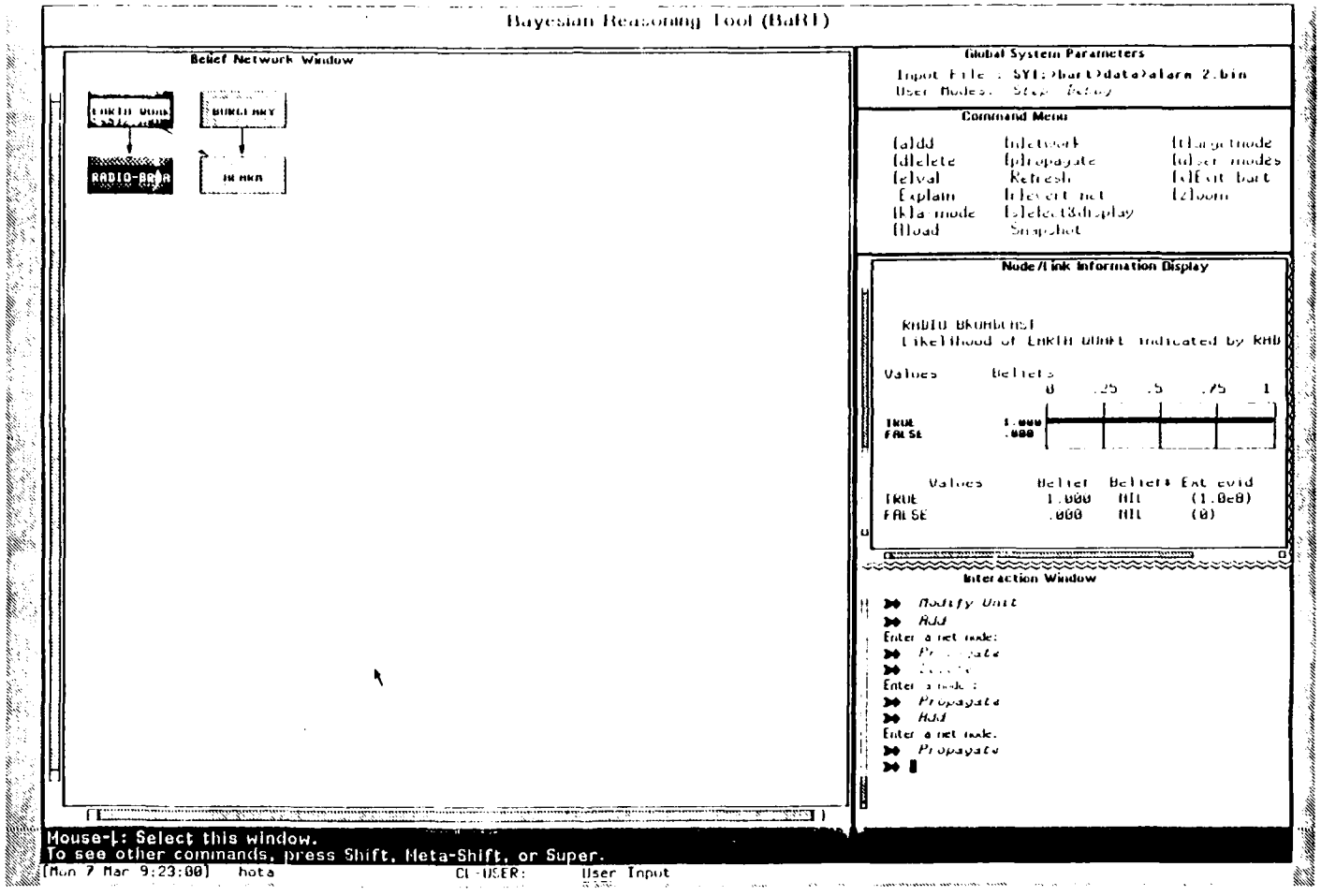

Figure 1 - Top-Level Screen Display

Top-Level BaRT Commands

The possible choices are:

add :

Adds new external evidence to a node. This is done by clicking on *add* first and then clicking on the node to which the user wants to add external evidence. Now, a menu appears which has the heading *New External Evidence* followed by the node's name. (A sample screen menu for input of external evidence is shown in Figure 2.) All of the values of the hypothesis are listed on the left hand side of this menu. A scale is presented across the top with gradations of evidential support ranging from "Rule out" to "Affirms." The area under the scale heading has a triangular marker for each value which is initially placed under the position indicating indifference (i.e. no evidential impact one way or the other). The user can indicate the level of support (from the new external evidence) for each value by placing the cursor in the appropriate position and then mouse-clicking left, causing the marker to be placed there. The user has the option of choosing several modes for entering the evidence. The default scale has *Discrete* intervals, and the marker will be placed under the closest interval gradation to where the user clicks. The user can choose to make the scale discrete or continuous by clicking right on *Discrete/Continuous Scale*. A menu appears with the options *Continuous Scale* and *Discrete Scale*. The user should click-right on one of these to change this option. The numbers listed on top of the scale can be shown or hidden by clicking right on *Show/Hide Scale*. A menu appears with the options *Show Scale* and *Hide Scale*. The user should click-right on one of these to select this option. The default is to show the scale. Finally, the user can choose to enter the evidence in the form of a list of numbers by clicking left on *Numeric Vector* and then typing in the list of numbers followed by a carriage return. This list should be of length equal to the rank of that node where each number in the list gives the impact of the evidence on the belief in the corresponding value (in order) of the hypothesis. These numbers are to be interpreted as components of a likelihood vector having the standard semantics: numbers greater than 1.0 indicate values supported by the evidence, numbers less than 1.0 (but non-negative) indicate values argued against by the evidence, and 1.0 indicates no evidential impact one way or the other. Note that the markers under the scale move to the appropriate places after this list is entered. Now, mouse-click left on either *done* to enter the new evidence into the system or *abort* to ignore the change. Once the new evidence is entered, the effect of that evidence can be propagated by clicking on *propagate*. After clicking on *propagate* or *refresh*, the node to which external evidence has been added will be shown in reversed video (white name on a dark background) in the Belief Network Window.

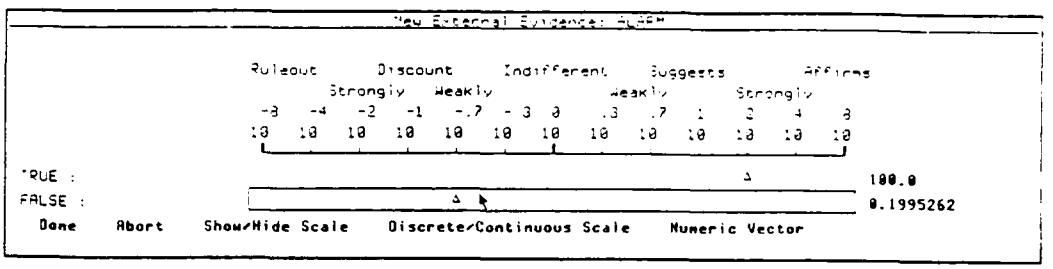

Figure 2 - External Evidence Menu

delete :

Deletes external evidence from a node. The user must click on *delete* and then click on the appropriate node. If there is only one piece of evidence for this node, that evidence will be deleted. If there are multiple pieces of evidence for this node, then a menu will pop up which lists each evidence and also the string "all." The user can click on the appropriate individual evidence he would like to delete or "all" to delete all evidence from this node. Note that a node which no longer has any evidence will stay in reversed video until the user clicks on *propagate* or *refresh.*

eval :

Clicking on this makes the LISP reader (displayed in the interaction window) read an expression and evaluate.

explain :

Explains the reasoning. Not yet implemented.

ka-mode :

Changes to the knowledge acquisition mode which allows the user to enter the information for a new network or change the information in an existing network. This will be described in full detail in the following section entitled *Knowledge Acquisition Mode.*

load :

Prompts for a data file and loads it. It performs all the necessary internal calculations, brings the network into equilibrium, and displays the network.

network :

Allows the user to change the current network to any previously loaded network. After clicking on *network*, a menu appears with a list of all previously loaded networks. The user should then click-left on the desired network.

propagate :

Updates the network and redisplays the information.

refresh :

Refreshes the display.

revert-net :

Resets the network back to the initial equilibrium state so the user can try a new run with new observations without loading and reinitializing.

select&display :

After choosing this, clicking on any node or link in the network displays the information about that object in the node/link display pane. Depending on which user modes have been selected, a histogram of the belief distribution for the values of a selected node may be displayed in addition to the actual belief and belief* vectors, the external evidence for the node, and a brief description of the node. The information displayed about links includes the λ, λ^*, π, and π^* vectors.

snapshot :

Saves the present environment in a file. Not yet implemented.

targetnode :

Indicates the influence of the other nodes in the network on this selected node. Not implemented in this version of the system.

user-modes :

Allows the user to set available options. After clicking on this, a temporary menu entitled *Select User Modes* of all the user setable options appears (See Figure 3). The user can change any of these values by clicking left on them. Presently nine global options appear with their present values in boldface. This command can be terminated by clicking either on *done* to process the request or *abort* to ignore the request. *Step mode* shows the propagation in steps, i.e., the system propagates one node at a time. This is useful if the user wants to see the results after each update of a node. Note that the user can tell whether the system has reached equilibrium: if the data file name in the global system parameters pane is in boldface, then the network is in equilibrium. *Debug mode* is not implemented at present. *Clear node/link window each time* determines whether the system will clear the node/link display pane before presenting new information or append the new information. The default is to clear the window each time. On the Sun, if the option is set to append and the buffer becomes full, then this system will automatically clear the old information and present the new information on a fresh window. *Update beliefs* determines whether the belief of the values of each hypothesis is updated when the system propagates the effect of new evidence. *Update belief** determines whether

the belief* vector of each hypothesis is updated when the system propagates the effect of new evidence. *Compute standard deviation* determines whether the standard deviation (of each value in the belief vector) of the hypotheses is updated when the system propagates the effect of new evidence. This supplies information concerning how much the belief in a particular value of the hypothesis might still vary. This computation is not correct for entire BaRT networks yet. It should only be used for the testing of a unit during knowledge acquisition. *Compute benefit factors* determines whether the measure of the entropy (uncertainty) in the belief values of each node in the network is updated when the system propagates the effect of new evidence. This influence is shown by the intensity of grayness in the Belief Network Window. *Display belief histogram* determines whether the belief histogram for a node is displayed in the node/link display window when a node is selected. *Maximum number of values to be displayed* allows the user to see the top *n* sorted values (based on belief) in the node/link display pane. To change this number, the user must click left on the current number. Then he must type in the new number and press *Return* (on the Symbolics it is preferable to press *End* rather than *Return*). The user can choose to display all of the belief values by typing a *0* as the number. The default is to display all of the belief values.

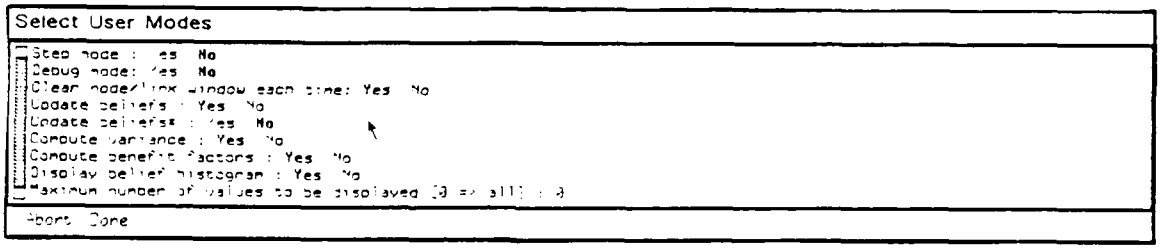

Figure 3 - User Modes Menu

exit-bart :

Exits from the program.

zoom :

Allows the nodes in the network to be increased or decreased in size. A menu pops up, and the user must click-left on the numeric field next to the label *zoom factor*. The user should type a positive number to increase the size, a negative number to decrease the size, or *0* to return to the original size. After typing the number, the user must press *return*. Then the user must click-left on *done* to process the request or on *abort* to ignore the request.

3.2.4. Knowledge Acquisition Mode

This version of BaRT provides a preliminary graphics interface for entering information about a network. The user must enter the nodes as a *unit*, where each unit contains a manifestation (child) node and all of its causes (parents) and the joint conditional probability tensor representing the relationship between the node and its parents. Note that the user does not need to specify links here. This definition of a unit was chosen because it allows one to focus on a local contained relationship; a node and all of its parents can be quantified locally by the joint conditional probability tensor in the BaRT model. The complete network will then be a full distribution composed of these local unit relationships.

Knowledge Acquisition Windows

After clicking on *ka-mode* in the top-level command menu, new windows will appear on the screen. At this state, the whole screen consists of six windows: the title pane, the cluster window, the belief network display, the knowledge acquisition command menu pane, the node/link information display pane, and the interaction window. Figure 4 provides a sample screen display.

The title window and the interaction window are the same as they were in the top level.

The cluster window does not display any information in the current implementation.

The belief network display is on the bottom left hand side of the screen occupying a large portion of the screen. This pane is used to display the current pieces of the network (subnetworks) which consist of nodes and their links.

The knowledge acquisition command menu pane is on the top right hand side and consists of the commands for knowledge acquisition. These are mouse sensitive and can be invoked by mouse-clicking left on them.

The node/link information display pane is below the command menu pane and is used to present information about nodes in the network.

Menu Commands for Knowledge Acquisition

The possible choices are:

add-evidence :

Adds external evidence to a node. The user must click on the node to which he wishes to add evidence. Then he must enter the evidence as a list of numbers of

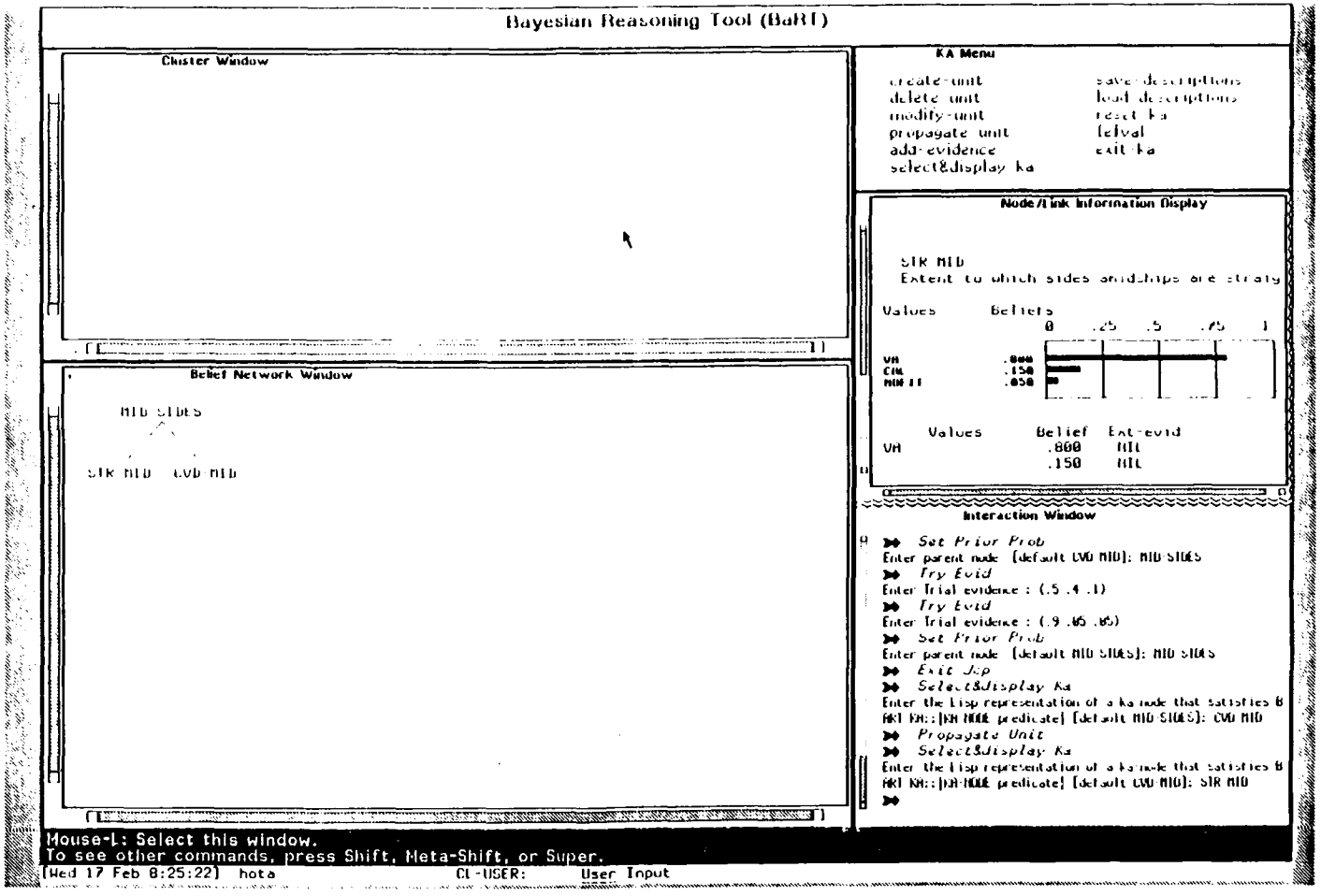

Figure 4 - Screen Display for Knowledge Acquisition Mode

length equal to the number of values of the hypothesis (where each entry corresponds to each value of the hypothesis in order) in the interaction window. These numbers are to be interpreted as components of a likelihood vector having the standard semantics: numbers greater than 1.0 indicate values supported by the evidence, numbers less than 1.0 (but non-negative) indicate values argued against by the evidence, and 1.0 indicates no evidential impact one way or the other. The user should then click on *propagate-unit* to see the effect of this evidence in the unit.

create-unit :

Creates a unit (a node and all of its parents, and the joint conditional probability tensor representing the relationship between the node and its parents). After clicking on *create-unit*, a menu pops up entitled *New KA Unit* with one line for the manifestation (child) node and one line for the first cause (parent) node. (See Figure 5.) The user must enter the information about the manifestation by clicking left on the field to the right of the words "Manifestation, list of values, its role." The user should enter a name for the node followed by a list of values and a documentation string in quotes. These three fields should be separated by commas. Then he should press *Return* and enter the causes similarly. The user can get this menu to scroll upward (using the scroll bar) to make room for additional causes to be entered. After all of the nodes for the unit are entered, the user should click on *done* to enter the unit or *abort* to ignore the request. After the user clicks on *done*, he will be put into the *Joint Conditional Probability* (JCP) mode. See the section entitled *Joint Conditional Probability Mode*.

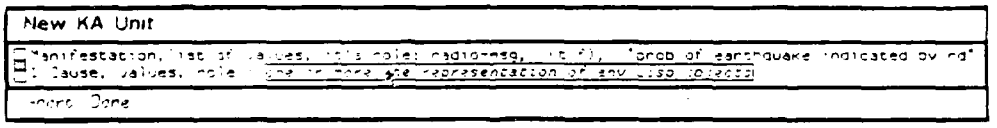

Figure 5 - Menu for Creating New Units

delete-unit :

Deletes a unit. This command deletes each node in the unit (a child node and all of its parent nodes) unless the node also belongs to another unit. The user must click on *delete-unit* and then click on the appropriate child node.

eval :

Clicking on this makes the LISP reader (displayed in the interaction window) read an expression and evaluate.

load-descriptions :

Prompts for a data file which was previously saved in knowledge acquisition format, loads it, and displays the network.

modify-unit :

Modifies a unit. After choosing this, the user must then click on a node of a unit. If a child node of a unit is selected, then a menu pops up with the choices *Add-a-parent*, *Delete-a-parent*, *Change-node-values*, or *Modify-JCP*, and the user must click on one of these. *Add-a-parent* will allow the user to add a parent node to this unit. A menu pops up for the addition of the one parent node. The user should enter the information as he would for *create-unit*. *Delete-a-parent* allows the user to delete a parent from this unit. The user must click on the specific parent he wants to delete. *Change-node-values* allows the user to change the values in the child (manifestation) node. A menu pops up which asks for the new list of values for this node. The user must click on the field labeled *a list* and then type in the new list of values. *Modify-JCP* allows the user to modify the joint conditional probability tensor for this unit. After any of the above choices, the user is placed in the JCP mode so the joint conditional probability tensor of this unit can be changed to reflect and remain consistent with the modification. If the user instead clicks on a top node of a subnetwork (one of the current pieces of the network) after clicking on *modify-unit*, then a menu pops up with the choices *Change-Nodes-Values* or *Change-Nodes-Prior*. For either choice, a menu pops up: the user must click on *a list* and type in the new list.

propagate-unit :

Brings the unit into equilibrium. If there is only one unit, then it will bring the unit into equilibrium. If there is more than one unit, then the system will ask which unit the user wants to propagate; the user must click on the child node of the unit he wants propagated. Note that the system will warn the user (in the interaction window) if he has not entered information in the joint conditional probability tensors or has not clicked on *done* in the JCP mode to enter the information. The unit will not propagate in this case.

reset-ka :

Clears everything away so the user can enter a new network.

save-descriptions :

Saves the current state of the network (or subnets) for use in BaRT or for later editing. If the current state is not acceptable for loading into the top level of BaRT (i.e., if there are several subnetworks rather than one combined network, or if some of the joint conditional probability tensors have not been entered), then the user will be prompted for a file name so the current state can be saved for future editing. Otherwise, a menu appears which asks the user whether he wants to save this in the current format for future editing or if he wants to save this for loading into the top level of BaRT. The user should click-left on the appropriate choice. In either case, the user is prompted for a file name. In the latter case, the user is also prompted for a network name.

select&display-ka :

After choosing this, clicking on any node in the network displays the information about that node in the node/link display pane. Depending on which user modes have been selected, a histogram of the belief distribution for the values of a selected node may be displayed in addition to the actual belief vectors, the external evidence for the node, and a brief description of the node.

exit-ka :

Exits from the knowledge acquisition phase and returns to the top-level command loop.

3.2.5. Joint Conditional Probability Mode

The user will enter this mode if he was creating or modifying a unit in knowledge acquisition mode. When the user is placed in this mode, he can modify the joint conditional probability tensor. The tensor is represented as a matrix in which the values of the child node are on the left and the combinations of the values of the parent nodes are listed across the top. Each entry represents the degree of belief that the child node would be equal to the value in that row given the combination of parent node values indicated at the top of that column. Note again that the numbers used to quantify the relationship do not have to be probabilities. All that is required is that the tensor entries are correct relative to each other. If the joint conditional probability matrix has just been created, the entries have question marks in them. If the matrix is not new, then it will contain numeric entries. These can be replaced with numbers as follows: the user must place the cursor over the entry he would like to change and then he must click-right while holding down the *Meta* and *Control* keys on the Symbolics. On the Sun, the user should click-left on this field. Then the user can type in the numeric value and press *Return*. The default for any entries not given by the user is "1.0." The user can now test the unit or return to the knowledge acquisition command level as described in the *JCP Menu Commands* section below.

Joint Conditional Probability Windows

When the user is placed in this mode, a new set of windows appears on the screen. At this state, the whole screen consists of six windows: the title pane, the joint conditional probability window, the belief network window, the JCP command menu pane, the node/link information display pane, and the interaction window. Figure 6 provides a sample screen display.

The title window, the belief network window, the node/link information display pane, and the interaction window are basically the same as they were in the knowledge acquisition level.

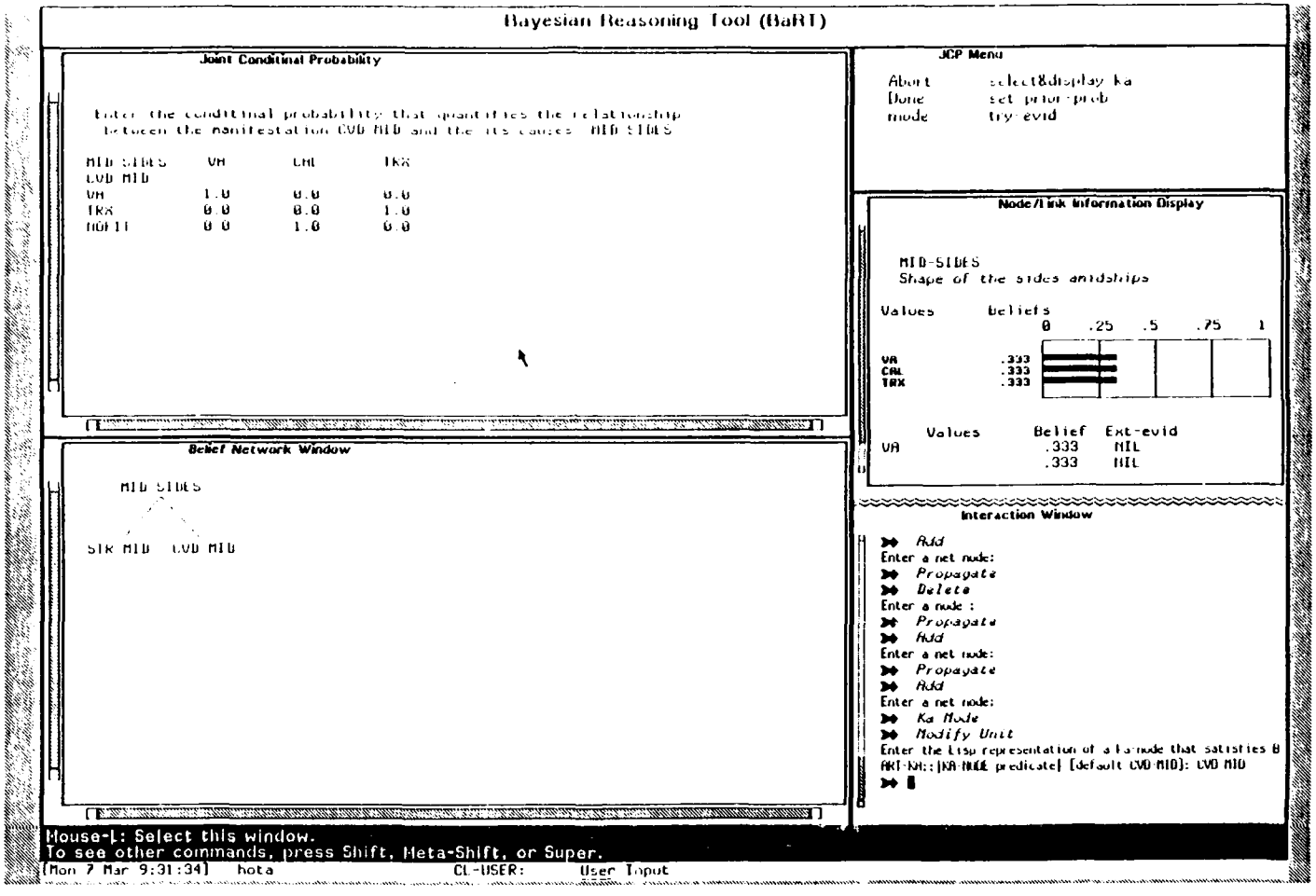

Figure 6 - Screen Display for Joint Conditional Probability Mode

The Joint Conditional Probability Window is used to display the current joint conditional probability tensor. Note that a unit was chosen or was being worked on in the knowledge acquisition level before the user is p. ced in JCP mode, so the current joint conditional probability tensor belongs to that particular unit. In order to modify another existing joint conditional probability tensor, the user must exit from the JCP mode and click on *modify-unit* in the Knowledge Acquisition menu.

The JCP command menu pane is on the top right hand side and consists of the commands for entering and testing the joint conditional probability tensor. These are mouse sensitive and can be invoked by mouse-clicking left on them.

JCP Menu Commands

abort :

Allows the user to exit from the JCP mode and return to the knowledge acquisition command level without entering any changes to the joint conditional probability tensor into the system.

done :

Allows the user to enter any changes to the joint conditional probability tensor into the system and exit from the JCP mode, returning to the knowledge acquisition command level. The user must click on this in order to propagate the unit later.

mode :

Allows the user to enter the joint conditional probability tensor using different methods. Not yet implemented.

select&display-ka :

After choosing this, clicking on any node in the network displays the information about that node in the node/link display pane. Depending on which user modes have been selected, a histogram of the belief distribution for the values of a selected node may be displayed in addition to the actual belief vectors, the external evidence for the node, and a brief description of the node.

set-prior-prob :

Allows the user to set or change the prior probability of a top node of the current unit selected. After selecting a parent node, a menu appears and the user must click-left on the field labeled *a list* to enter the new list of prior probabilities. Note that these values will stay as the list of prior probabilities for this node even after exiting the JCP mode. However, they will not be used in the propagation of a unit in the knowledge acquisition command level unless they are the prior probabilities for a top node of a subnetwork.

try-evid :

Allows the user to add temporary evidence to the child node of the unit being worked on. A prompt is given in the Interaction Window to enter the list of evidence. This evidence is propagated through the unit and the new belief of the child node is displayed. Note that this evidence will not stay with the node once the user exits the JCP mode.

3.2.6. Avoiding Potential Problems While Running BaRT

On the Symbolics, it is preferable for the user to press the *End* key rather than the *Return* key to enter information in BaRT windows/menus. This will help to avoid display problems.

If there is a problem or error in LISP while on the Sun, the user should not type :a to abort to the LISP top level because this will bring the user out of the BaRT LISP top level and into Sun's Lucid LISP top level. The user should instead type the command *(back)* to get back to the top level within BaRT.

3.3. Example

This example was presented in Kim[4]. Say an alarm in a house rings when there is an intrusion or when there is an earthquake. Also, earthquakes are reported on the radio. The nodes and links and their prior probabilities and conditional probabilities for this problem are given below as they would be written in a data file. The network is shown pictorially in Figure 7. (The information for this network is in the data file called alarm.lisp.)

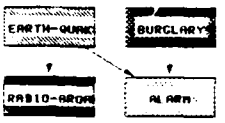

Figure 7 - Alarm Network

;;;-========== Beginning of data file -========

;;;-======= network information -=====
(make-net 'alarm-network)

```
;;;-=-=-=-=-=-= node information -=-=-=-=

(make-node
  'ALARM
  :role                    "Probability of ALARM ringing"
  :condpro1                '((0.76 0.24) (0.73 0.27) (0.28 0.72) (0.19 0.81))
  :parents-ord             '(BURGLARY EARTHQUAKE))
;;; where
;;; Probability [ALARM=true | BURGLARY=true and EARTHQUAKE=true] = 0.76
;;; Probability [ALARM=false | BURGLARY=true and EARTHQUAKE=true] = 0.24
;;; Probability [ALARM=true | BURGLARY=true and EARTHQUAKE=false] = 0.73
;;; Probability [ALARM=false | BURGLARY=true and EARTHQUAKE=false] = 0.27
;;; Probability [ALARM=true | BURGLARY=false and EARTHQUAKE=true] = 0.28
;;; Probability [ALARM=false | BURGLARY=false and EARTHQUAKE=true] = 0.72
;;; Probability [ALARM=true | BURGLARY=false and EARTHQUAKE=false] = 0.19
;;; Probability [ALARM=false | BURGLARY=false and EARTHQUAKE=false] = 0.81


(make-node
  'BURGLARY
  :role                    "Probability of BURGLARY"
  :prior                   '(0.1 0.9))


(make-node
  'EARTHQUAKE
  :role                    "Probability of EARTHQUAKE "
  :prior                   '(0.2 0.8))


(make-node
  'RADIO-BROADCAST
  :role                    "Probability of EARTHQUAKE
                             indicated by RADIO-BROADCAST "
  :condpro1                '((0.8 0.2) (0.001 0.999)))
;;; where
;;; Probability [RADIO-BROADCAST=true | EARTHQUAKE=true] = 0.8
;;; Probability [RADIO-BROADCAST=false | EARTHQUAKE=true] = 0.2
;;; Probability [RADIO-BROADCAST=true | EARTHQUAKE=false] = 0.001
;;; Probability[ RADIO-BROADCAST=false | EARTHQUAKE=false] = 0.999
```

```
;;; -======= link information -=======

(make-link
  'lk-BURGLARY->ALARM
  :tnode                'BURGLARY
  :bnode                'ALARM)


(make-link
  'lk-EARTHQUAKE->ALARM
  :tnode                'EARTHQUAKE
  :bnode                'ALARM)


(make-link
  'lk-EARTHQUAKE->RADIO-BROADCAST
  :tnode                'EARTHQUAKE
  :bnode                'RADIO-BROADCAST)

;;;-======== End of file -========
```

To run BaRT with this data file, do the following:

Invoke BaRT as described in Section 3.1. Click on *load*. This prompts for the data file name. Type the data file name. The program brings the network into equilibrium. At this stage we do not have any external evidence, so information on any node displays nil in the external evidence field. Now click on *add* and then click on the node *alarm*. Change the external evidence of that node by clicking under *Affirms* in the row labeled *True* (to show that it is true that the alarm is ringing). Note that the marker in the row labeled *False* automatically moves to the position *Ruleout* (to show that it is therefore false that the alarm is not ringing). (See Figure 8.) Once the evidence is given, click on *propagate*. This propagates the new evidence in the network, changing the beliefs of the nodes, and brings the network into equilibrium. Click on *select&display* and then click on a node or a link to see information about that node or link in the node/link display window. The evidence that the alarm is ringing would actually increase the belief of both burglary and earthquake. Now change the external evidence fields of radio broadcast to *Affirms* in the row labeled *True*. Again, this automatically moves the marker to *Ruleout* in the row labeled *False*. This change would result in a further increase in the belief of earthquake and reduces the belief of burglary.

To run the program again with the initial settings, click on *revert-net* and proceed.

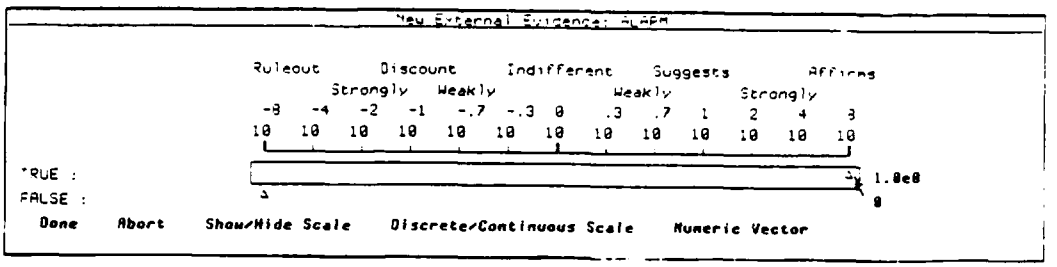

Figure 8 - Entering Evidence for Alarm Network

4. Implementation

BaRT is implemented in PCL on top of Common LISP. A graphic interface is provided on Symbolics and Suns showing the network (the nodes and their relations), the belief vectors, the belief* vectors, the coefficients, and the dynamic propagation of beliefs (or belief*) as new evidence is obtained.

Several generic classes of objects are defined for BaRT. Instances of networks are the individual networks which can be loaded. Instances of network nodes represent hypotheses of the domain and instances of network links represent the relationship between hypotheses.

Each network node carries the following information: a vector containing the possible values which the hypothesis can hold, a vector which stores the belief distribution of that node, and a vector which stores the belief* distribution used for the categorical assertion of that node. A topmost node (one which has no parents) also has a vector containing the prior probabilities of the values of the hypothesis. The number of elements in the belief, belief*, and prior probability vectors is equal to the number of values that the hypothesis can take (the rank), and each element represents the belief/belief*/prior probability of the corresponding element in the values vector. All network nodes except topmost nodes also carry a tensor containing the joint conditional probabilities which represent the relationship between the node (manifestation) and all of its parents (causes). (See Appendix A for a definition of *tensor*.) The joint conditional probability tensor is provided by the user as described in the sections on *Data Files* and *Knowledge Acquisition*. This tensor is of order (n + 1) with indices ijk..l where i is the rank of the node, j,k...l are the ranks of its parents, and n is the total number of parents. This is represented as a list of lists (a matrix) of the form

$$((P[X_1 \mid U_{1_1}\, U_{2_1} \ldots U_{n_1}]\; P[X_2 \mid U_{1_1}\, U_{2_1} \ldots U_{n_1}] \ldots\; P[X_m \mid U_{1_1}\, U_{2_1} \ldots U_{n_1}])$$
$$(P[X_1 \mid U_{1_1}\, U_{2_1} \ldots U_{n_2}]\; P[X_2 \mid U_{1_1}\, U_{2_1} \ldots U_{n_2}] \ldots\; P[X_m \mid U_{1_1}\, U_{2_1} \ldots U_{n_2}])$$
$$\ldots\ldots$$
$$(P[X_1 \mid U_{1_p}\, U_{2_r} \ldots U_{n_r}]\; P[X_2 \mid U_{1_p}\, U_{2_r} \ldots U_{n_r}] \ldots\; P[X_m \mid U_{1_p}\, U_{2_r} \ldots U_{n_r}]))$$

where X is a node and U_1, U_2, ..., U_n are all of its parents,

$P[X_i \mid U_{1_j}, U_{2_k}, \ldots, U_{n_l}]$ is the probability of $X = X_i$ given that $U_1 = U_{1_j}$, $U_2 = U_{2_k}$, ..., $U_n = U_{n_l}$,

X_1, X_2, ..., X_m are the possible values of X,

U_{1_1}, U_{1_2}, ... U_{1_p}, are the possible values of U_1,

U_{2_1}, U_{2_2}, ..., U_{2_r}, are the possible values of U_2, ...,

and U_{n_1}, U_{n_2}, ..., U_{n_r}, are the possible values of U_n.

Each network link connects two nodes (a causal or top node of the link and a manifestation or bottom node of the link) and carries the following information: a vector λ which indicates the diagnostic support from the bottom node, a vector π which indicates the causal support from the top node, a vector λ^* which represents the diagnostic support for the belief* distribution, and a vector π^* which represents the causal support for the belief* distribution. λ, λ^*, π and π^* are all of degree n where n is the rank of the causal node.

Each node has procedures attached to it. *Update* is one such procedure that can update the belief and/or belief* of a node. When new evidence is obtained for a network node, it can be propagated in the network by changing the incoming λ (or λ^*) or π (or π^*) at the node (λ (or λ^*) in the case of a manifestation, π (or π^*) in the case of a cause). The system immediately detects the introduced inconsistency (i.e., the difference between the old coefficients and the current ones) and updates the node by calculating the new belief (or belief*) vector using all the incoming λs (or λ^*s), πs (or π^*s) and the joint conditional probability tensor. The new coefficients which will be sent to neighboring nodes are also calculated. Now, if the new coefficients sent to neighboring nodes are different from the old coefficients, then these nodes are updated also. This propagation continues until there is no further change in the coefficients, and the network reaches equilibrium. This procedure is described in more detail in the following paragraphs.

In each update of the belief of a node, two variables, effective λ and effective π of the node, are calculated using all incoming coefficients (the λs and πs of all the links connected to the node). Effective λ is the term product of all the incoming λs. Effective π is the tensor product of the combined conditional probability tensor and the outer product of all incoming πs. Then the ratio of the belief is calculated as the term product of the effective λ and the effective π. Absolute belief is obtained by normalizing this ratio with respect to 1. After calculating the belief, updating involves calculating the new

coefficients (πs and λs) for all the links of the node. The new coefficient of a link is the belief of the node supported by all the incoming πs and λs except that particular λ or π of the link for which the new coefficient is being calculated. New πs of a link are calculated by taking the term quotient of the new belief of the node and the incoming λ of that link. The new λ of a link is the matrix product of the term product of all λs and the tensor product of the combined conditional probability tensor and the outer product of all the πs except that particular π that is associated with the link for which the new λ is being calculated. Once these new coefficients are calculated, the updating procedure involves comparing these new coefficients against the old ones and finding out which link's coefficients are changed. If a change is detected, then the neighboring nodes at the other end of these links must be updated. The new nodes are updated, and this propagation continues until there is no further change in the coefficients, and equilibrium is reached. The equations for updating belief are shown in Appendix A.

The computation of belief* is slightly different from belief updating. In belief updating, individual support from all of the node's neighbors is added whereas in belief* updating, these individual supports are maximized. As shown in the equations in Appendix A, this can be achieved by replacing the first operator + used in the inner product by the operator *max* and replacing all πs with π*s and λs with λ*s. A fuller discussion of the belief and belief* updating equations used in BaRT is given in Booker [2]. For more details refer to Pearl[3,5,6,7].

All the basic procedures that have been introduced so far allow one to update the network and bring it into equilibrium. All of the core functions except the knowledge acquisition functions are defined in a package called *bart* which is in the file named *bart*. This does not have any interface and can be used on any machine with Common LISP and PCL. The user can call some of the functions in this file from another program; these functions are described in Appendix C. The knowledge acquisition core functions are defined in a package called *bart-ka* which is in the file named *bart-ka*. These should be used with the graphic interface on the Symbolics and Suns. A non-graphic user interface is developed for use on any machine with Common LISP and PCL (see Appendix B); this code is in the file *bart-frame-tty*. The graphic user interface (machine dependent) is developed for two machines: Symbolics and Suns. The interface code for the Symbolics is in the file *bart-frame-3600-ka* and for the Sun it is in *bart-frame-sun-ka*. Anything a particular programmer wants to add can be added here in *bart-frame-<machine>*. The name of the package in the user interface files is *bart-frame*. All the general utilities are in the package *bart-util* which resides in the file *bart-util*. Presently, four sample data files are being used and they are in the directory called *data* in the bart directory.

5. Selected Implementation Details

The following paragraphs discuss some of the implementation details which affect the system's efficiency.

• The update procedure can be invoked recursively to update all of the nodes. However, this is inefficient because of the large stack spaces that it would have to maintain. Instead we use a global procedure which first updates a node and then places the nodes that are returned on a list so it can later update these new nodes. This way the recursive overhead is avoided. Choosing which node to update or, alternatively, where to place the freshly returned to-be-updated nodes in the global list can also affect the program's efficiency. When placing a node in the global list, it is moved to the end of the list if it was already in there. Since the updating procedure takes one node at a time from the beginning of this global list, the updating of an affected node is postponed as long as possible, resulting in fewer updates to reach equilibrium when multiple evidence is available at the same time.

• The coefficients λ and π represent their support as a ratio. After finding the new values of each coefficient during the updating of a node, these coefficients are normalized before they are stored at the appropriate links. Since the old and new values of the coefficients are compared to determine whether there has been a change, comparing the normalized values allows the system to avoid duplicate updates, and the system can attain equilibrium more quickly.

• The coefficients are real numbers and are represented as floating numbers. Comparing these real numbers using the built-in "equal" function compares them to the last decimal digit (16th). Instead, assuming a small error, and comparing the numbers to some nth (4,5...) digit reduces the computation greatly without any significant affect on the accuracy. The present implementation compares numbers up to the 4th digit. This can be changed by changing the value of the global variable *precision* to any desired accuracy. The default value of this variable is .0001.

• Calculating new πs at a node involves two inner products, one outer product and one term product for each π for each iteration. Inner products are costly. Instead the product

$$\left(\mathbf{P} \bullet \mathbf{\Lambda}\right) \ast \mathbf{\Pi}$$

where
$\ast$ is the term product
• is the inner product
P is the joint conditional probability tensor
Λ is the term product of incoming λs
Π is the outer product of incoming πs (these terms are described more fully in

is calculated for each iteration and then πs are calculated by summing over different indices of the above product. This saves a great deal of execution time by reducing multiplications and divisions to additions.

Appendix A

Tensor Product Computation

A *tensor* is a mathematical object that is a generalization of a vector to higher orders. The order of a tensor is the number of indices needed to specify an element. A vector is therefore a tensor of order one and a matrix is a tensor of order two. Three standard operations defined on tensors are relevant to this discussion:

Term Product The term product is defined between two tensors **A** and **B** having the same indices. Each element in the resulting tensor **C** is simply the product of the elements with the corresponding indices from **A** and **B**.

$$\mathbf{C} = \mathbf{A} * \mathbf{B} \quad \text{where} \quad c_{i_1 \cdots i_n} = a_{i_1 \cdots i_n} \times b_{i_1 \cdots i_n}$$

Outer Product The outer product of two tensors **A** and **B** having order m and n respectively is a tensor **C** of order $m+n$. Each element of **C** is the product of the elements of **A** and **B** whose aggregate indices correspond to its own indices.

$$\mathbf{C} = \mathbf{A} \circ \mathbf{B} \quad \text{where} \quad c_{i_1 \cdots i_m j_1 \cdots j_n} = a_{i_1 \cdots i_m} \times b_{j_1 \cdots j_n}$$

Inner Product The inner product of two tensors **A** and **B** is a tensor formed by taking the outer product of **A** and **B** and then summing up over common indices that appear both in **A** and **B**. If **A** is of order m, **B** is of order n and they have k common indices then the inner product **C** is a tensor of order $(m-k)+(n-k)$.

$$\mathbf{C} = \mathbf{A} \bullet \mathbf{B} \quad \text{where} \quad c_{i_1 \cdots i_{m-k} j_1 \cdots j_{n-k}} = \sum_{l_1, \cdots, l_k} a_{i_1 \cdots i_{m-k} l_1 \cdots l_k} \times b_{l_1 \cdots l_k j_1 \cdots j_{n-k}}$$

Equations

Let λ_{Y_j}, $\lambda^*_{Y_j}$, π_{U_i}, and $\pi^*_{U_i}$ be vectors (or, equivalently, tensors of order 1) whose elements are the messages a node X receives from its children and its parents respectively:

$$\lambda_{Y_j} = \left(\lambda_{Y_j}(x_1), \cdots, \lambda_{Y_j}(x_r)\right) \quad \text{where } r \text{ is the number of possible values for } X$$

$$\lambda^*_{Y_j} = \left(\lambda^*_{Y_j}(x_1), \cdots, \lambda^*_{Y_j}(x_r)\right) \quad \text{where } r \text{ is the number of possible values for } X$$

$$\pi_{U_i} = \left(\pi_X(u_{i_1}), \cdots, \pi_X(u_{i_{r(i)}})\right) \quad \text{where } r(i) \text{ is the number of possible values for } U_i$$

$$\pi^*_{U_i} = \left(\pi^*_X(u_{i_1}), \cdots, \pi^*_X(u_{i_{r(i)}})\right) \quad \text{where } r(i) \text{ is the number of possible values for } U_i$$

The term product of all λ_{Y_j} vectors is another vector Λ of length r given by

$$\Lambda = \lambda_{Y_1} * \cdots * \lambda_{Y_m} = \left(\prod_{j=1}^{m} \lambda_{Y_j}(x_1), \cdots, \prod_{j=1}^{m} \lambda_{Y_j}(x_r)\right)$$

The term product of all $\lambda^*_{Y_j}$ vectors is another vector Λ^* of length r given by

$$\Lambda^* = \lambda^*_{Y_1} * \cdots * \lambda^*_{Y_m} = \left(\prod_{j=1}^{m} \lambda^*_{Y_j}(x_1), \cdots, \prod_{j=1}^{m} \lambda^*_{Y_j}(x_r)\right)$$

The outer product of all π_{U_i} vectors is a tensor Π of order n given by

$$\Pi = \pi_{U_1} \circ \cdots \circ \pi_{U_n} \text{ where } \pi_{k_1 \cdots k_n} = \prod_{i=1}^{n} \pi_X(u_{i_k})$$

The outer product of all π'_{U_i} vectors is a tensor Π' of order n given by

$$\Pi' = \pi'_{U_1} \circ \cdots \circ \pi'_{U_n} \text{ where } \pi'_{k_1 \cdots k_n} = \prod_{i=1}^{n} \pi'_X(u_{i_k})$$

We can consider the set of fixed probabilities $P(x \mid u_1, \cdots, u_n)$ as elements of a tensor $\mathbf{P}$ of order $n+1$. Now if we compute the inner product of $\mathbf{P}$ with Π we obtain a tensor of order 1 (the indices for the U_i are common to both tensors):

$$\mathbf{P} \bullet \Pi = \left(\sum_{i_1, \cdots, i_n} P(x_1 \mid u_{i_1}, \cdots, u_{i_n}) \prod_{k=1}^{n} \pi_X(u_{i_k}) , \cdots , \sum_{i_1, \cdots, i_n} P(x_r \mid u_{i_1}, \cdots, u_{i_n}) \prod_{k=1}^{n} \pi_X(u_{i_k}) \right)$$

If we make the summation operator explicit, we can rewrite the formula as

$$\mathbf{P} \bullet_+ \Pi = \left(\sum_{i_1, \cdots, i_n} P(x_1 \mid u_{i_1}, \cdots, u_{i_n}) \prod_{k=1}^{n} \pi_X(u_{i_k}) , \cdots , \sum_{i_1, \cdots, i_n} P(x_r \mid u_{i_1}, \cdots, u_{i_n}) \prod_{k=1}^{n} \pi_X(u_{i_k}) \right)$$

We can now denote the formula for BEL as

$$\mathbf{BEL} = \alpha \, \Lambda \ast (\mathbf{P} \bullet_+ \Pi)$$

If we compute the inner product of $\mathbf{P}$ with Π' we obtain a tensor of order 1:

$$\mathbf{P} \bullet \Pi' = \left(\sum_{i_1, \cdots, i_n} P(x_1 \mid u_{i_1}, \cdots, u_{i_n}) \prod_{k=1}^{n} \pi'_X(u_{i_k}) , \cdots , \sum_{i_1, \cdots, i_n} P(x_r \mid u_{i_1}, \cdots, u_{i_n}) \prod_{k=1}^{n} \pi'_X(u_{i_k}) \right)$$

The *BEL'* computation requires us to maximize over all elements u_k rather than taking a sum, so we can redefine the inner product operator as a maximize operator, and we can denote this new inner product with the symbol $\bullet_{\max}$.

$$\mathbf{P} \bullet_{\max} \Pi' = \left(\max_{i_1, \cdots, i_n} P(x_1 \mid u_{i_1}, \cdots, u_{i_n}) \prod_{k=1}^{n} \pi'_X(u_{i_k}) , \cdots , \max_{i_1, \cdots, i_n} P(x_r \mid u_{i_1}, \cdots, u_{i_n}) \prod_{k=1}^{n} \pi'_X(u_{i_k}) \right)$$

Now the *BEL'*(x) computation can be written in tensor notation as

$$\mathbf{BEL}' = \alpha \, \Lambda' \ast (\mathbf{P} \bullet_{\max} \Pi')$$

Moreover, it is clear that we can use similar methods to compute the messages that node X will send to its neighbors. The vector π_{Y_j} destined for child Y_j can be computed by term-by-term division of the elements of **BEL** by the elements of λ_{Y_j}, and the vector π'_{Y_j} can be computed by term-by-term division of the elements of **BEL'** by the elements of λ'_{Y_j}. The vector λ_X destined for parent U_i can be computed just like **BEL** except that we replace the vector π_{U_i} with a unit vector $(1, \cdots, 1)$ of equal length when computing the outer product Π, and the vector λ'_X can be computed just like **BEL'** except that we replace the vector π'_{U_i} with a unit vector $(1, \cdots, 1)$ of equal length when computing the outer product Π'.

The beliefs and belief commitments can be computed in one uniform scheme as shown in Figure 9.

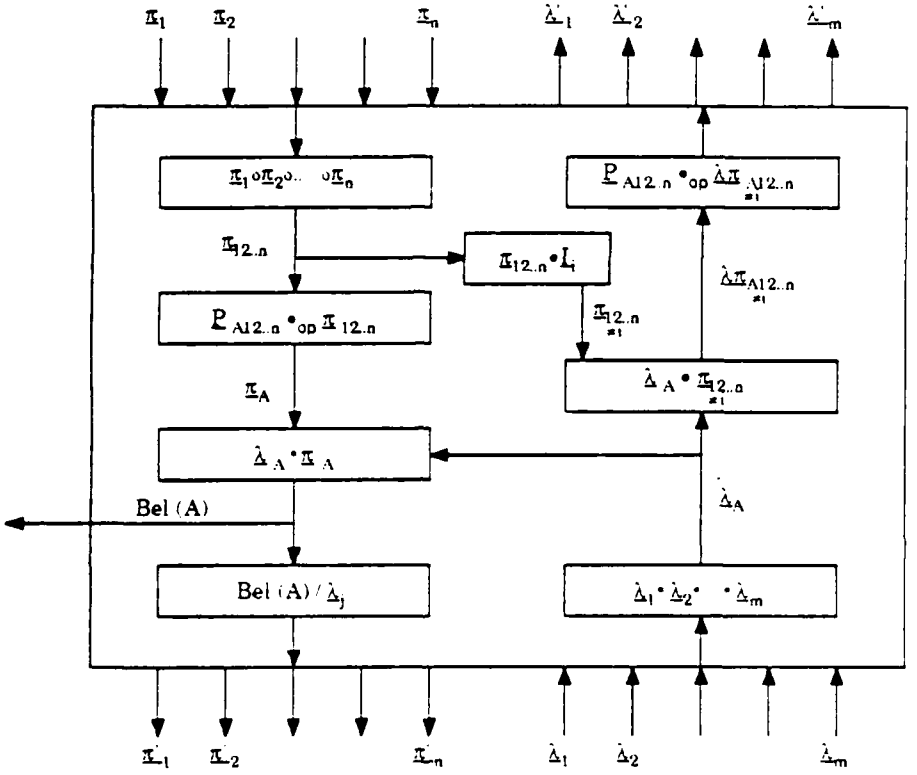

Figure 9. Combined updating of belief and belief commitment (the operator $\bullet_{op}$ represents the standard or modified inner product depending on whether op is $\cdot$ or max). In belief commitment all the πs, λs, and Bels should be changed to π^*s, λ^*s, and Bel*s.

Appendix B - Using the System without the Graphic Interface

BaRT can be run without the graphic interface. The user must get into a Common LISP environment which supports PCL. Load the files *bart-defsys.lisp*, *bart-util*, *bart*, and *bart-frame-tty* (in order) which are in the *src* subdirectory of the *bart* directory. If the user is on a terminal other than a Symbolics or a Sun, then he can just type the command *(bart-util::load-bart)* instead of explicitly loading the files named above. Now, type the command *(in-package 'bart-frame)*.

After loading a data file using *gen-load*, the following LISP functions can be invoked:

gen-add :

Adds new external evidence to a node. The arguments for this function are a node name, a list representing new evidence, and a network name. The node name is an optional argument. If a node name is not specified, this function will prompt for one. If an invalid node name is given, a list of the valid node names will appear, and the function will again prompt for a node name. The function will then prompt for a list of numbers (of length equal to the rank of that node) representing the new evidence; each number in the list gives the impact of the evidence on the belief in the corresponding value (in order) of the hypothesis. If new evidence for a particular value is unknown, that can be given by including a "1.0" in the list corresponding to that value. If a network name is not given in the list of parameters, it always defaults to the current network. The effect of the new evidence can be propagated by calling *gen-propagate*.

gen-delete :

Deletes external evidence from a node. This function takes a node name and a network name as optional arguments. If a node name is not specified, this function will prompt for one. If an invalid node name is given, a list of the valid node names will appear, and the function will again prompt for a node name. If a network name is not given in the list of parameters, it always defaults to the current network. If the specified node only has one piece of evidence, then this will be deleted. If there are more than one, then a list of the evidences will be presented so the user can choose which evidence he wants deleted; the user will also be given the choice to delete all of the evidences. The effect of the removal of the evidence can be propagated by calling *gen-propagate*.

gen-display-nodes :

Displays information about the nodes. This function takes a node or a list of nodes and a network as optional arguments. If one node is given as the first argument, this function displays only the information about that node. If a list of nodes is

given as the first argument, this function displays the information about the nodes in the list. If no argument is given, this function displays the information about all of the nodes. If a network name is not given in the list of parameters, it always defaults to the current network.

gen-display-links :

Displays information about the links. This function takes a link or a list of links and a network name as optional arguments. If one link is given as the first argument, this function displays only the information about that link. If a list of links is given as the first argument, this function displays the information about the links in the list. If no argument is given, this function displays the information about all of the links. If a network name is not given in the list of parameters, it always defaults to the current network.

gen-load :

Loads a data file. This function takes a file name as an optional argument and loads the file. If a file name is not specified, then this function will prompt for one. It performs the necessary internal calculations and then brings the network into equilibrium. An example of the gen-load command on the Symbolics might be *(gen-load "local:>bart>data>ship.lisp")* and an example of this command on the Sun might be *(gen-load "/usr/prj/bart/data/ship.lisp")*.

gen-network :

Allows the user to switch back and forth between networks which are already loaded. This function takes a network name as an optional argument. If a network name is not specified, this function will prompt for one.

gen-propagate :

Updates the network. This function takes a network name as an optional argument. If a network name is not given, it always defaults to the current network.

gen-revert-net :

Resets the network back to the initial equilibrium state so the user can try a new run with new observations without loading and reinitializing. This function takes a network name as an optional argument. If a network name is not given, it always defaults to the current network.

gen-select&display :

Displays the information about a node or a link. This function takes a node or a link name and a network name as optional arguments. If a node or a link name is not specified, this function will prompt for one. If an invalid name is given, a list of the valid node and link names will appear, and the function will again prompt for a name. If a network name is not given in the list of parameters, it always

defaults to the current network.

In addition to the above functions, the user has the ability to change the value of *bart::*compute-bel*-p** which determines whether or not the belief* values are updated when the network is propagated. The user should type

(setf (bart::*compute-bel*-p* bart::*sys-state*) t)

to update the belief* values and

(setf (bart::*compute-bel*-p* bart::*sys-state*) nil)

to stop updating the belief* values. The default is not to update these values.

Appendix C - BaRT Functions Which Can be Called from Another Program

Certain BaRT functions can be called from other programs as long as the user is in a Common LISP environment which supports PCL. These functions reside in the package *bart* which is in the file *bart*. The user must load the files *bart-defsys.lisp*, *bart-util* and *bart* (in order) which are in the *src* subdirectory of the *bart* directory.

All functions (except those pointed out below) return *true* unless the user calls a BaRT function with an incorrect argument. In that case, a list of n elements containing error information is returned. The user can process this list however is most convenient for him. The first element in the list is the atom *&err&*. The second element is an integer which can be decoded as follows:

1 – The given network <arg> is illegal
2 – New evidence given <arg> is not a list
3 – All the elements in the given new evidence <arg> are not numbers
4 – Length of the new evidence supplied <arg1> is not equal to the rank of the node <arg2>
5 – New evidence is not supplied for the node <arg>
6 – Illegal node name <arg>
7 – File <arg> doesn't exist
8 – Initial equilibrium has not yet been reached for the net <arg>
9 – Illegal list of node names <arg>
10 – Illegal list of link names <arg>
11 – Illegal link name <arg>
12 – Illegal list of object names <arg>
13 – Illegal object name <arg>
14 – Illegal evidence node/link
15 – node <arg> has more than 1 evidence
16 – no evidence present for <arg> node

The third through (n - 1)th elements are objects related to the error message. The last (nth) element is a string which states the error message.

The following BaRT functions can be called from other programs:

int-gen-add :

Adds new external evidence to a node. The arguments for this function are a node name, a list representing new evidence (of length equal to the rank of that node), and a network name. Each element in the evidence list gives the impact of the evidence on the belief in the corresponding value (in order) of the hypothesis. If new evidence for a particular value is unknown, that can be given by including a "1.0"

in the list corresponding to that value. The network name is an optional argument, and if it is not given in the list of parameters, it always defaults to the current network. The effect of the new evidence can be propagated by calling *int-gen-propagate*.

int-gen-delete :

Deletes external evidence from a node. The arguments for this function are an object name and a network name. The network name is an optional argument, and if it is not given in the list of parameters, it always defaults to the current network. The object argument can be the network node which the user wants to delete evidence from, the lambda link which connects the evidence node to the network node, or the evidence node the user wishes to delete. If the argument is a network node and this network node has more than one evidence node, then an error message will be returned. The effect of the removal of the evidence can be propagated by calling *int-gen-propagate*. Note that if the user wishes to see the list of evidence nodes, he can retrieve them with the function *int-gen-get-evid-nodes*.

int-gen-display-nodes :

Displays information about the nodes. This function takes a stream, a node or a list of nodes, and a network name as arguments. The network name is an optional argument, and if it is not given in the list of parameters, it always defaults to the current network. If one node is given as the second argument, this function displays only the information about that node. If a list of nodes is given as the second argument, this function displays the information about the nodes in the list. If the second argument is nil or if no second argument is given, this function displays the information about all of the nodes.

int-gen-display-links :

Displays information about the links. This function takes a stream, a link or a list of links, and a network name as arguments. The network name is an optional argument, and if it is not given in the list of parameters, it always defaults to the current network. If one link is given as the second argument, this function displays only the information about that link. If a list of links is given as the second argument, this function displays the information about the links in the list. If the second argument is nil or if no second argument is given, this function displays the information about all of the links.

int-gen-get-evid-nodes :

Returns all of the evidence nodes for the given network node as long as the arguments are correct or an error message if the arguments are incorrect. The arguments for this function are a node name and a network name. The network name is an optional argument, and if it is not given in the list of parameters, it always

defaults to the current network. The evidence is returned as a list of sublists where each sublist contains the evidence node name and a list of the evidence values for that node. The information returned by this function is useful when the user wishes to delete an evidence node.

int-gen-load :

Loads a data file. This function takes a file name as an argument and loads the file. It performs the necessary internal calculations and then brings the network into equilibrium.

int-gen-network :

Allows the user to switch back and forth between networks which are already loaded. This function takes a network name as an argument.

int-gen-propagate :

Updates the network. This function takes a network name as an optional argument. If a network name is not given, it always defaults to the current network.

int-gen-revert-net :

Resets the network back to the initial equilibrium state so the user can try a new run with new observations without loading and reinitializing. This function takes a network name as an optional argument. If a network name is not given, it always defaults to the current network.

int-gen-select&display :

Displays the information about a node or a link. This function takes a stream, a node or a link name, and a network name as arguments. If a network name is not given in the list of parameters, it always defaults to the current network.

get-ptr :

Returns the internal pointer to the node or the link. This function takes a node or a link name and a network name as arguments. If a network name is not given in the list of parameters, it always defaults to the current network. Note that this macro returns only the internal pointer or nil; it does not return an error message if the arguments are incorrect.

get-ptr-i :

Returns the internal pointer to the node or the link. This function takes a node or a link name and an internal pointer to a network as arguments. If the internal network name is not given in the list of parameters, it always defaults to the current network. Note that this macro returns only the internal pointer or nil; it does not return an error message if the arguments are incorrect.

In addition to the above functions, the user has the ability to change the value of *bart::*compute-bel*-p** which determines whether or not the belief* values are updated when the network is propagated. The user should type

(setf (bart::*compute-bel*-p* bart::*sys-state*) t)

to update the belief* values and

(setf (bart::*compute-bel*-p* bart::*sys-state*) nil)

to stop updating the belief* values. The default is not to update these values.

References

1. Booker, L. B. and Hota, N., Probabilistic Reasoning About Ship Images. Proceedings of the 2nd AAAI Workshop on Uncertainty in Artificial Intelligence. Philadelphia, PA. August 8-10. 1986. p.29-36; also to appear in *Uncertainty in Artificial Intelligence*, Lemmer and Kanal (Eds.), North Holland, 1987.

2. Booker, L. B., Hota, N. and Hemphill, G., Implementing a Bayesian Scheme for Revising Belief Commitments. Proceedings of the 3rd AAAI Workshop on Uncertainty in Artificial Intelligence, Seattle, WA, July 10-12, 1987. p.348-354.

3. Geffner, H. and Pearl, J., Distributed Diagnosis of Systems with Multiple Faults. Technical Report CSD-860023, Computer Science Department, University of California, Los Angeles, CA, December 1986.

4. Kim, J., CONVINCE: A **CONV**ersational **IN**ference **C**onsolidation **E**ngine. Ph.D. Dissertation, University of California, Los Angeles, 1983.

5. Kim, J. and Pearl, J., A Computation Model for Combined Causal and Diagnostic Reasoning in Inference Systems. *Proceedings of IJCAI-83*, Los Angeles, CA, August 1983. p.190-193.

6. Pearl, J., Fusion, Propagation, and Structuring in Belief Networks. *Artificial Intelligence*, Vol 9, p.241-288, 1986.

7. Pearl, J., Distributed Revision of Belief Commitment in Multi-Hypotheses Interpretation. Proceedings of the 2nd AAAI Workshop on Uncertainty in Artificial Intelligence. Philadelphia, PA. August 8-10. 1986. p.201-209.

8. Shafer, G., Tversky, A., Languages and Designs for Probability Judgement. Cognitive Science, Vol 9, p.309-339, 1985.

END
DATE
9-88
DTIC